

# JDBC - EXCEPTIONS HANDLING

<http://www.tutorialspoint.com/jdbc/jdbc-exceptions.htm>

Copyright © tutorialspoint.com

Exception handling allows you to handle exceptional conditions such as program-defined errors in a controlled fashion.

When an exception condition occurs, an exception is thrown. The term thrown means that current program execution stops, and control is redirected to the nearest applicable catch clause. If no applicable catch clause exists, then the program's execution ends.

JDBC Exception handling is very similar to Java Exception handling but for JDBC, the most common exception you'll deal with is **java.sql.SQLException**.

## SQLException Methods:

A SQLException can occur both in the driver and the database. When such an exception occurs, an object of type SQLException will be passed to the catch clause.

The passed SQLException object has the following methods available for retrieving additional information about the exception:

Method	Description
getErrorCode( )	Gets the error number associated with the exception.
getMessage( )	Gets the JDBC driver's error message for an error handled by the driver or gets the Oracle error number and message for a database error.
getSQLState( )	Gets the XOPEN SQLstate string. For a JDBC driver error, no useful information is returned from this method. For a database error, the five-digit XOPEN SQLstate code is returned. This method can return null.
getNextException( )	Gets the next Exception object in the exception chain.
printStackTrace( )	Prints the current exception, or throwable, and its backtrace to a standard error stream.
printStackTrace(PrintStream s)	Prints this throwable and its backtrace to the print stream you specify.
printStackTrace(PrintWriter w)	Prints this throwable and its backtrace to the print writer you specify.

By utilizing the information available from the Exception object, you can catch an exception and continue your program appropriately. Here is the general form of a try block:

```
try {  
    // Your risky code goes between these curly braces!!!  
}  
catch(Exception ex) {  
    // Your exception handling code goes between these  
    // curly braces, similar to the exception clause
```

```

    // in a PL/SQL block.
}
finally {
    // Your must-always-be-executed code goes between these
    // curly braces. Like closing database connection.
}

```

## Example:

Study the following example code to understand the usage of **try....catch...finally** blocks.

```

//STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
    // JDBC driver name and database URL
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    static final String DB_URL = "jdbc:mysql://localhost/EMP";

    // Database credentials
    static final String USER = "username";
    static final String PASS = "password";

    public static void main(String[] args) {
        Connection conn = null;
        try{
            //STEP 2: Register JDBC driver
            Class.forName("com.mysql.jdbc.Driver");

            //STEP 3: Open a connection
            System.out.println("Connecting to database...");
            conn = DriverManager.getConnection(DB_URL,USER,PASS);

            //STEP 4: Execute a query
            System.out.println("Creating statement...");
            Statement stmt = conn.createStatement();
            String sql;
            sql = "SELECT id, first, last, age FROM Employees";
            ResultSet rs = stmt.executeQuery(sql);

            //STEP 5: Extract data from result set
            while(rs.next()){
                //Retrieve by column name
                int id  = rs.getInt("id");
                int age = rs.getInt("age");
                String first = rs.getString("first");
                String last = rs.getString("last");

                //Display values
                System.out.print("ID: " + id);
                System.out.print(", Age: " + age);
                System.out.print(", First: " + first);
                System.out.println(", Last: " + last);
            }
            //STEP 6: Clean-up environment
            rs.close();
            stmt.close();
            conn.close();
        }catch(SQLException se){
            //Handle errors for JDBC
            se.printStackTrace();
        }catch(Exception e){
            //Handle errors for Class.forName
            e.printStackTrace();
        }finally{
            //finally block used to close resources
            try{
                if(conn!=null)
                    conn.close();
            }catch(SQLException se){
                se.printStackTrace();
            }
        }
    }
}

```

```
        } //end finally try
    } //end try
    System.out.println("Goodbye!");
} //end main
} //end JDBCExample
```

Now let us compile above example as follows:

```
C:\>javac JDBCExample.java
C:\>
```

When you run **JDBCExample**, it produces following result if there is no problem, otherwise corresponding error would be caught and error message would be displayed:

```
C:\>java JDBCExample
Connecting to database...
Creating statement...
ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
C:\>
```

Try above example by passing wrong database name or wrong username or password and check the result.